Tutorblatt Aufgabe 3: Erweiterung BNE

Ziel der Aufgabe ist es, das Steuerwerk so zu erweitern, sodass ein Branch Not Equal Befehl (BNE) ausgeführt werden kann. Dabei vergleicht der BNE-Befehl die zwei Quellregister. Falls diese nicht gleich sind, wird ein relativer Offset auf den Programm Counter (PC) addiert.

Um die Aufgabe zu lösen, sehen wir uns zunächst nochmals an wie der Branch-Equal (BEQ)-Befehl in einem Single-Cycle-RISC-V-Prozessor funktioniert.

Die Branch-Equal-Instruktion entspricht dem B-Typ-Instruktionsformat. Das B in B-Typ steht für Branch und dieses Instruktionsformat wird von allen Instruktionen, welche bedingte Sprünge implementieren, umgesetzt.

Der B-Typ ist folgendermaßen aufgebaut: Zuerst kommen das 12. und das 10. bis 5. Bit des Immediates. Der Immediate ist hierbei der Wert, welchen wir im Sprungfall auf den PC addiert wird. Danach folgen das zweite, sowie das erste Quellregister, welche verglichen werden sollen. Beide Register werden jeweils mit 5 Bit adressiert. Dann kommt der 3-Bit Funktionscode, welche die Instruktion innerhalb des Opcodes nochmals spezifizieren. Danach folgen das 4. - 1. Bit, sowie das 11 bit des immediates. Und am Ende steht der 7-bit Opcode, welche bei den Branch-Befehlen, die wir in der Vorlesung definiert haben, stets den Wert 99 hat.

Zur Ausführung wird zunächst die Branch-Instruktion aus dem Instruktionsspeicher geladen und der Befehl dekodiert. Die Control-Unit liest die Bits 0 – 6, also den Opcode 99. Daraufhin wird der funct3-code ausgelesen, welcher beim branch-equal auf 000 steht. Der funct7-code wird von der control-unit hier ignoriert, da eine B-Type-Instruktion, wie vorhin gesehen, keinen funct7-code besitzt.
Weiters werden die zwei Quellregister aus der Instruktion extrahiert und deren Werte aus dem Registerfile gelesen. Hierbei wird auch noch versucht, ein drittes Register zu extrahieren, welches in anderen Befehlen dazu verwendet wird, das Ergebnis einer Berechnung in einem Register zu speichern. Da dies allerdings bei unserem Befehl nicht nötig ist, setzt die Control-Unit das Write-Enable-Bit des Register-Files auf 0, sodass keine ungewollten Änderungen im Registerfile erfolgen.
Als letztes wird dann noch der Immediate aus der Instruktion extrahiert. Die Control-Unit setzt dabei das Immediate-Soure Signal auf 10, sodass die korrekten Bits als Immediate interpretiert werden. Der Immediate wird auf den aktuellen PC addiert, um die Adresse zu berechnen, zu welcher möglicherweise gesprungen wird. Parallel dazu wird auch die Adresse von der nächsten Instruktion berechnet, falls der Sprung nicht genommen wird, also aktueller PC + 4.
Es wurden also die zwei spezifizierten Quellregister ausgelesen, deren Werte jetzt zur ALU propagiert werden. Dabei setzt die Control-Unit das ALU-Src Bit auf 0, da wir ja den Register-Inhalt des zweiten Quellregisters in der ALU haben wollen, und nicht den Immediate. Zudem spezifiziert die Control-Unit mit dem ALUControl-Bit, dass die zwei Registerwerte voneinander subtrahiert werden sollen. Falls das Ergebnis der Subtraktion gleich 0 ist, waren die beiden Werte offensichtlich gleich, was die ALU mithilfe des Zero-Flags an die Control-Unit zurückliefert. Da das Branch-Bit bei unseren Branch-Befehl natürlich auf 1 gesetzt ist

und bei Gleichheit das Zero-Bit auch, wird durch die UND-Verknüpfung das PCSrc-Bit auf 1 gesetzt. Somit ist die nächste Adresse des Programmcounters nicht die aktuelle Adresse + 4 (also 32 Bit), sondern die aktuelle Adresse + der sign-extendete Immediate. Dadurch wird ein relativer Sprung realisiert. Falls die Subtraktion nicht 0 ergeben hat, waren die Register nicht gleich, weshalb das Zero-Flag von der ALU auf 0 gesetzt wird, wodurch die UND-Verknüpfung mit dem Branch-Bit wiederum 0 ergibt, wodurch das PCSrc auf 0 bleibt. Somit wäre die nächste Adresse des Programmcounters die aktuelle Adresse + 4, es wird also ganz normal im Programmfluss die nächste Instruktion abgearbeitet.

Da wir nun wissen, wie eine Branch-Equal-Instruktion abgearbeitet wird, können wir uns überlegen, welche Gatter zur Control-Unit hinzugefügt werden müssen, um einen Branch-Not-Equal-Befehl umzusetzen.
Dazu schauen wir uns zunächst mal die zwei Befehle in der Instruktionstabelle an. Wie wir erkennen können, unterscheiden sich die beiden Instruktionen lediglich am letzten funct3-Bit. Dieses ist beim BEQ auf 0 gesetzt und beim BNE auf 1 gesetzt. Also wissen wir bereits, dass wir das Verhalten der Control-Unit von diesem Bit abhängig machen müssen.
Das Zero-Signal sagt uns beim BEQ, ob die zwei Register gleich sind. Wenn es also auf 0 gesetzt ist, sagt es uns also, dass die zwei Register ungleich sind – genau die Information, die wir benötigen.

Wir verwenden also das Zero-Flag und das 0te Bit von funct3, um die gewünschte Funktionalität umzusetzen. Dazu unterschieden wir die 4 möglichen Fälle.

- Falls funct3 = 0 und zero = 0, dann handelt es sich um eine BEQ-Instruktion. Allerdings sind die Register ungleich, weshalb wir nicht springen möchten.
- Falls funct3 = 0 und zero = 1, dann handelt es sich um eine BEQ-Instruktion, wobei die Register gleich sind, also ist hier ein Sprung erwünscht.
- Falls funct3 = 1 und zero = 0, dann handelt es sich um eine BNE-Instruktion, wobei die Register ungleich sind, also ist hier ein Sprung erwünscht.
- Falls funct3 = 1 und zero = 1, dann handelt es sich um eine BNE-Instruktion. Allerdings sind die Register gleich, weshalb wir nicht springen möchten.

Wir können klar erkennen, dass unsere gewünschte Funktionalität einem einfachen XOR-Gatter entspricht. Deshalb legen wir das 0te funct3-Bit und das Zero-Bit an einem XOR-Gatter an, dessen Ergebnis wir mit dem Branch-Signal ver-UNDen. Da sich die Ausführung des BNE-Befehls von der Ausführung des BEQ-Befehls nur in der Interpretation des Vergleich-Ergebnisses unterschiedet, läuft die Ausführung, bis auf unsere kleine Änderung genau so ab, wie bei einem BEQ-Befehl – was bedeutet, dass wir keine weiteren Änderungen am Single-Cycle-Prozessor vornehmen müssen. Also haben wir mit möglichst wenig Aufwand und möglichst viel Wiederverwendung von bestehender Funktionalität, die gefragte Instruktion implementiert.